



UNIVERSIDADE FEDERAL DA BAHIA
INSTITUTO DE COMPUTAÇÃO
MATA60 - Banco de Dados

BRENO LEONARDO LIMA MACEDO
LUIZ CLÁUDIO DANTAS CAVALCANTI

II AVALIAÇÃO PRÁTICA: GRUPO K

Salvador
2023

II AVALIAÇÃO PRÁTICA: GRUPO K

Arquivos do trabalho: [Google Drive](#)


Repositório do GitHub da implementação: [GitHub](#)

Vídeo Demonstração: [Vídeo Demonstração](#)

Mudanças no mapeamento:

Após a entrega da I Avaliação Prática, decidimos fazer algumas modificações no nosso banco de dados:

1. Em *tipo_atividade*, adicionamos a coluna *requer_supervisor* como tipo **boolean**.



A screenshot of a database table schema for 'tipo_atividade'. The table has the following columns and attributes:

id	integer	PK	AK
nome	varchar(255)		
tipo_carga_horaria	char(1)		
limite_horas	integer		
horas	integer	NULL	
requer_supervisor	boolean		
codigo_curso	integer		FK

2. Em *solicitacao_aproveitamento*, os campos *nome_supervisor*, *tel_supervisor* e *email_supervisor* não são mais **NOT NULL**. Essa mudança é uma consequência da mudança 1. Além disso, adicionamos novos campos *resposta_coordenador* (**varchar** que será **NULL** até que a solicitação seja avaliada e uma justificativa seja fornecida caso seja rejeitada) e *data_da_solicitacao* (tipo **date** que é por padrão a data atual no momento da inserção da solicitação na tabela).

solicitacao_aproveitamento				
id	integer	PK	AK	
descricao	varchar(4000)			
resposta_coordenador	varchar(4000)	NULL		
data_da_solicitacao	date			
situacao	varchar(255)			
carga_real	integer			
carga_aproveitada	integer	NULL		
nome_supervisor	varchar(255)	NULL		
tel_supervisor	varchar(20)	NULL		
email_supervisor	varchar(255)	NULL		
matricula_aluno	integer			FK
matricula_coordenador	integer	NULL		FK
id_tipo	integer			FK

Normalizações:

Relações a serem avaliadas:

As seguintes relações fazem parte do banco de dados do nosso trabalho. As chaves primárias estão sublinhadas e as chaves estrangeiras estão em itálico.

Curso(codigo, nome, horas_extensao, horas_gerais, *matricula_coordenador*)

Tipo_atividade(id, nome, tipo_carga_horaria, limite_horas, horas, requer_supervisor, *codigo_curso*)

Aluno(matricula, cpf, telefone, nome, email, hash_senha, *codigo_curso*)

Coordenador(*matricula_siape*, nome, hash_senha, email)

Solicitacao_aproveitamento(id, descricao, resposta_coordenador, data_da_solicitacao, situacao, carga_real, carga_aproveitada, nome_supervisor, tel_supervisor, email_supervisor, *matricula_aluno*, *matricula_coordenador*, *id_tipo*)

Anexo(num, *solicitacao_id*, nome, extensao, caminho)

Primeira Forma Normal:

Constatamos que o atributo *nome* em *Aluno* e *Coordenador* (pensado como o atributo que receberia o nome completo do usuário) ficaria melhor decomposto e atômico, em ambas as tabelas, se fosse substituído por *nome* e *sobrenome*. O mesmo pode ser feito para *nome_supervisor* em *Solicitacao_aproveitamento*, sendo decomposto em *nome_supervisor* e *sobrenome_supervisor*.

Não há outros atributos que não sejam atômicos, não há atributos multivalorados, e não há relações aninhadas.

Após os ajustes necessários, as tabelas *Aluno*, *Coordenador* e *Solicitacao_aproveitamento* ficaram da seguinte forma:

Aluno(matricula, cpf, telefone, nome, sobrenome, email, hash_senha, *codigo_curso*)

Coordenador(matricula_siape, nome, sobrenome, hash_senha, email)

Solicitacao_aproveitamento(id, descricao, resposta_coordenador, data_da_solicitacao, situacao, carga_real, carga_aproveitada, nome_supervisor, sobrenome_supervisor, tel_supervisor, email_supervisor, *matricula_aluno*, *matricula_coordenador*, *id_tipo*)

2ª Forma Normal:

Considerando a versão adaptada à 1FN, não é necessário fazer mais modificações para adequar as relações à 2FN, pois todos os atributos não-primos de qualquer relação são totalmente dependentes da chave primária da mesma relação.

3ª Forma Normal:

Na relação *Solicitacao_aproveitamento*, embora não seja impossível que dois supervisores tenham o mesmo *nome_supervisor* e *sobrenome_supervisor*, na

esmagadora maioria dos casos, os atributos *email_supervisor*, *tel_supervisor* dependem de *nome_supervisor* e *sobrenome_supervisor* juntos. Portanto, existe a dependência funcional transitiva:

1. $id \rightarrow nome_supervisor, sobrenome_supervisor$
2. $nome_supervisor, sobrenome_supervisor \rightarrow email_supervisor, tel_supervisor$
3. $id \rightarrow email_supervisor, tel_supervisor$

Esta dependência funcional transitiva viola a 3FN. Adequando o banco de dados à Terceira Forma Normal, temos:

Solicitacao_aproveitamento(id, descricao, resposta_coordenador, data_da_solicitacao, situacao, carga_real, carga_aproveitada, *matricula_aluno*, *matricula_coordenador*, *id_tipo*, *id_supervisor*)

Supervisor(id, nome, sobrenome, email, telefone)

Optamos por adicionar a chave primária *id* à nova relação *Supervisor* porque, ainda que se aplique à vasta maioria dos casos, não queremos restringir um par <nome, sobrenome> a um único supervisor.

Forma Normal de Boyce-Codd:

Em nenhuma das relações há atributo que tenha dependência funcional de um outro atributo que não seja superchave.

Esclarecendo um caso específico, apesar da relação *Anexo* ter dois atributos que compõem sua chave primária, nem o atributo *num* nem o atributo *solicitacao_id* são UNIQUE. Os outros atributos de *Anexo* não têm dependência funcional com nenhum dos dois individualmente, apenas com a chave primária como um todo.

Portanto, as relações estão na Forma Normal de Boyce-Codd.

Resultado da normalização:

Curso(codigo, nome, horas_extensao, horas_gerais, *matricula_coordenador*)

Tipo_atividade(id, nome, tipo_carga_horaria, limite_horas, horas, *codigo_curso*)

Aluno(matricula, cpf, telefone, nome, sobrenome, email, hash_senha,

codigo_curso)

Coordenador(matricula_siape, nome, sobrenome, hash_senha, email)

Solicitacao_aproveitamento(id, descricao, resposta_coordenador,
data_da_solicitacao, situacao, carga_real,
carga_aproveitada, *matricula_aluno*,
matricula_coordenador, *id_tipo*, *id_supervisor*)

Supervisor(id, nome, sobrenome, email, telefone)

Anexo(num, solicitacao_id, nome, extensao, caminho)

Mudanças no mapeamento após normalização:

Após a normalização e após discussões entre os membros do grupo, as seguintes mudanças no mapeamento foram necessárias:

1. Na tabela *aluno*, adicionamos a coluna *sobrenome* do tipo varchar.

aluno			
matricula	integer	PK	AK
cpf	char(11)		AK
telefone	varchar(20)	NULL	
nome	varchar(255)		
sobrenome	varchar(255)		
email	varchar(255)		AK
hash_senha	char(64)		
codigo_curso	integer		FK

2. Na tabela *coordenador*, adicionamos a coluna *sobrenome* do tipo varchar.

matricula_siape	integer	PK	AK
nome	varchar(255)		
sobrenome	varchar(255)		
hash_senha	char(64)		
email	varchar(255)		AK

3. Na tabela *solicitacao_aproveitamento*, removemos as colunas *nome_supervisor*, *tel_supervisor*, *email_supervisor*, e adicionamos a chave estrangeira *id_supervisor*, podendo ser **NULL**.

id	integer	PK	AK
descricao	varchar(4000)		
resposta_coordenador	varchar(4000)	NULL	
data_da_solicitacao	date		
situacao	varchar(255)		
carga_real	integer		
carga_aproveitada	integer	NULL	
matricula_aluno	integer		FK
matricula_coordenador	integer	NULL	FK
id_tipo	integer		FK
id_supervisor	integer		FK

3.1. Criamos a tabela *supervisor* para representar a entidade composta pelos dados que ficavam nas colunas *nome_supervisor*, *sobrenome_supervisor*, *tel_supervisor* e *email_supervisor* de *solicitacao_aproveitamento*. Decidimos que a coluna *sobrenome* pode ser **NULL** caso o supervisor informado seja uma pessoa jurídica. Já com as normalizações e também com a coluna adicional *id* que é chave primária, temos:

id	integer	PK
email	varchar(255)	
nome	varchar(255)	
sobrenome	varchar(255)	NULL
telefone	varchar(20)	

Por fim, a versão atual do nosso mapeamento é a seguinte:

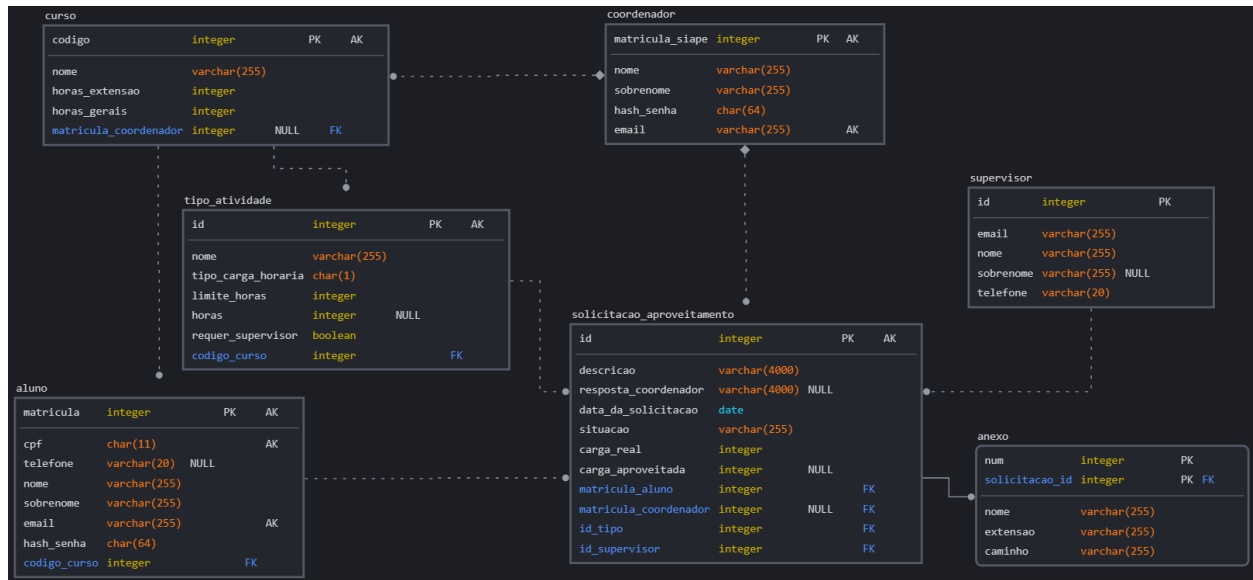
LEGENDA:

PK: chave primária

FK: chave estrangeira

AK: chave candidata (UNIQUE)

NULL: anulável (caso ausente, o campo é estritamente NOT NULL)



A imagem em alta resolução pode ser acessada [nesse link](#).

Mudanças nas tabelas no SQL:

1. Na tabela *coordenador*, adicionamos a coluna *sobrenome* como proposto na normalização.

```
CREATE TABLE coordenador
(
    matricula_siape          integer PRIMARY KEY,
    nome                     varchar(255) NOT NULL,
    sobrenome                 varchar(255) NOT NULL,
    hash_senha                char(64) NOT NULL,
    email                     varchar(255) UNIQUE NOT NULL
);
```

2. Na tabela *aluno*, também adicionamos a coluna *sobrenome* como proposto na normalização.

```
CREATE TABLE aluno (
    matricula                integer PRIMARY KEY,
```



```

cpf                char(11) UNIQUE NOT NULL,
telefone           varchar(20),
nome               varchar(255) NOT NULL,
sobrenome           varchar(255) NOT NULL,
email              varchar(255) UNIQUE NOT NULL,
hash_senha         char(64) NOT NULL,
codigo_curso       integer NOT NULL REFERENCES curso(codigo)
);

```

3. Na tabela *solicitacao_aproveitamento*, removemos as colunas *nome_supervisor*, *tel_supervisor* e *email_supervisor*, e adicionamos a coluna *id_supervisor* como proposto na normalização. Além disso, adicionamos as colunas *resposta_coordenador* e *data_da_solicitacao*.

```

CREATE TABLE solicitacao_aproveitamento (
  id                serial PRIMARY KEY,
  descricao         varchar(4000) NOT NULL,
  resposta_coordenador varchar(4000),
  data_da_solicitacao date NOT NULL DEFAULT now(),
  situacao          varchar(20) NOT NULL,
  carga_real        integer NOT NULL,
  carga_aproveitada integer,
  matricula_aluno    integer NOT NULL REFERENCES aluno(matricula),
  matricula_coordenador integer REFERENCES
coordenador(matricula_siape),
  id_tipo           integer NOT NULL REFERENCES
tipo_atividade(id),
  id_supervisor      integer REFERENCES supervisor(id)
);

```

4. Criação da tabela *supervisor*.

```

CREATE TABLE supervisor
(
  id                serial PRIMARY KEY,
  email             varchar(255) NOT NULL,
  nome              varchar(255) NOT NULL,
  sobrenome         varchar(255),
  telefone          varchar(20) NOT NULL
);

```

5. Adição da coluna *requer_supervisor* em *tipo_atividade*.

```
CREATE TABLE tipo_atividade (  
  id                serial PRIMARY KEY,  
  nome              varchar(255) NOT NULL,  
  tipo_carga_horaria char(1) NOT NULL DEFAULT 'G',  
  limite_horas      integer NOT NULL,  
  horas             integer,  
  requer_supervisor boolean NOT NULL DEFAULT false,  
  codigo_curso       integer NOT NULL REFERENCES curso(codigo)  
);
```

A criação de todas as tabelas em SQL está disponível no arquivo [create_sem_indices.sql](#).

Mudanças nas consultas:

1. Incluir a coluna *data_da_solicitacao*, *descricao*, *carga_real*, *carga_aproveitada*, *resposta_coordenador*, *id_supervisor*, *A.sobrenome* no resultado, e usar “AS” na coluna *T.nome* do resultado para evitar colisão com *A.nome*. Além disso, adicionamos o *order by* para ordenar pelas solicitações mais recentes.

```
SELECT S.id, S.situacao, S.data_da_solicitacao, S.descricao, S.carga_real,  
S.carga_aproveitada, S.resposta_coordenador, S.id_supervisor, T.nome AS  
"nome_atividade", T.tipo_carga_horaria, T.requer_supervisor, T.horas,  
T.limite_horas, A.matricula, A.nome, A.sobrenome FROM  
solicitacao_aproveitamento AS S INNER JOIN tipo_atividade AS T ON  
S.id_tipo=T.id INNER JOIN aluno AS A ON S.matricula_aluno=A.matricula  
INNER JOIN curso AS C ON C.codigo=T.codigo_curso WHERE  
C.matricula_coordenador=@matricula order by S.data_da_solicitacao desc;
```

2. Incluir a coluna *data_da_solicitacao*, *descricao*, *carga_real*, *carga_aproveitada*, *resposta_coordenador*, *id_supervisor*, *A.sobrenome* no resultado, e usar “AS” na coluna *T.nome* do resultado para evitar colisão com *A.nome*. Além disso, adicionamos o *order by* para ordenar pelas solicitações mais recentes.

```
SELECT S.id, S.situacao, S.data_da_solicitacao, S.descricao, S.carga_real,  
S.carga_aproveitada, S.resposta_coordenador, S.id_supervisor, T.nome AS  
"nome_atividade", T.tipo_carga_horaria, T.requer_supervisor, T.horas,  
T.limite_horas, A.matricula, A.nome, A.sobrenome FROM  
solicitacao_aproveitamento AS S INNER JOIN tipo_atividade AS T ON  
S.id_tipo=T.id INNER JOIN aluno AS A ON S.matricula_aluno=A.matricula
```

```
INNER JOIN curso AS C ON C.codigo=T.codigo_curso WHERE
C.matricula_coordenador=@matricula AND S.situacao=@situacao order by
S.data_da_solicitacao desc;
```

3. Incluir a coluna `data_da_solicitacao`, `descricao`, `carga_real`, `carga_aproveitada`, `resposta_coordenador`, `id_supervisor`, `A.sobrenome` no resultado, e usar "AS" na coluna `T.nome` do resultado para evitar colisão com `A.nome`. Além disso, adicionamos o `order by` para ordenar pelas solicitações mais recentes.

```
SELECT S.id, S.situacao, S.data_da_solicitacao, S.descricao, S.carga_real,
S.carga_aproveitada, S.resposta_coordenador, S.id_supervisor, T.nome AS
"nome_atividade", T.tipo_carga_horaria, T.requer_supervisor, T.horas,
T.limite_horas, A.matricula, A.nome, A.sobrenome FROM
solicitacao_aproveitamento AS S INNER JOIN tipo_atividade AS T ON
S.id_tipo=T.id INNER JOIN aluno AS A ON S.matricula_aluno=A.matricula
INNER JOIN curso AS C ON C.codigo=T.codigo_curso WHERE
matricula_aluno=@matricula order by S.data_da_solicitacao desc;
```

4. Incluir a coluna `data_da_solicitacao`, `descricao`, `carga_real`, `carga_aproveitada`, `resposta_coordenador`, `id_supervisor`, `A.sobrenome` no resultado, e usar "AS" na coluna `T.nome` do resultado para evitar colisão com `A.nome`. Além disso, adicionamos o `order by` para ordenar pelas solicitações mais recentes.

```
SELECT S.id, S.situacao, S.data_da_solicitacao, S.descricao, S.carga_real,
S.carga_aproveitada, S.resposta_coordenador, S.id_supervisor, T.nome AS
"nome_atividade", T.tipo_carga_horaria, T.requer_supervisor, T.horas,
T.limite_horas, A.matricula, A.nome, A.sobrenome FROM
solicitacao_aproveitamento AS S INNER JOIN tipo_atividade AS T ON
S.id_tipo=T.id INNER JOIN aluno AS A ON S.matricula_aluno=A.matricula
INNER JOIN curso AS C ON C.codigo=T.codigo_curso WHERE
matricula_aluno=@matricula AND situacao=@situacao order by
S.data_da_solicitacao desc;
```

7. Fazer INNER JOIN com as tabelas *supervisor*, *tipo_atividade*, *aluno*, e LEFT OUTER JOIN com *coordenador*, e selecionar manualmente os campos que serão incluídos.

```
SELECT SA.*, S.nome AS nome_supervisor, S.sobrenome AS
sobrenome_supervisor, S.email AS email_supervisor, S.telefone AS
tel_supervisor, T.nome AS nome_atividade, T.tipo_carga_horaria,
T.limite_horas, T.horas, A.nome AS nome_aluno, A.sobrenome AS
sobrenome_aluno, C.nome AS nome_coordenador, C.sobrenome AS
```

```
sobrenome_coordenador FROM solicitacao_aproveitamento AS SA INNER JOIN
supervisor AS S ON SA.id_supervisor=S.id INNER JOIN tipo_atividade AS T ON
SA.id_tipo=T.id INNER JOIN aluno AS A ON SA.matricula_aluno=A.matricula
LEFT OUTER JOIN coordenador AS C ON
SA.matricula_coordenador=C.matricula_siape WHERE SA.id=@solicitacao;
```

9. Incluir a coluna *requer_supervisor* no resultado.

```
SELECT
    id,
    nome,
    tipo_carga_horaria,
    limite_horas,
    horas,
    requer_supervisor
FROM
    tipo_atividade
WHERE
    codigo_curso = @curso;
```

13. Correção na explicação: @horas_complementares deve ser substituído por *horas_extensao* ou *horas_gerais*, de acordo com o valor de @carga desejado.

14. Incluir a coluna *requer_coordenador* no resultado.

```
SELECT
    id,
    nome,
    tipo_carga_horaria,
    limite_horas,
    horas,
    requer_supervisor
FROM
    tipo_atividade
WHERE
    codigo_curso = @curso
    AND tipo_carga_horaria = @carga;
```

15. Corrige para considerar o limite de horas e retornar o que resta, e não o total atual.

```
SELECT
```

```

    T.limite_horas - SUM(carga_aproveitada) AS horas_restantes
FROM
    solicitacao_aproveitamento AS S INNER JOIN tipo_atividade AS T ON
    S.id_tipo = T.id
WHERE
    matricula_aluno = @aluno
    AND S.id_tipo = @id_tipo
    AND S.situacao = 'Aprovada';

```

Novas Queries

16. Pensada para ser usada na tela de criação de solicitação, para que o aluno escolha um tipo de atividade no qual não tenha atingido o limite de horas. Retorna a soma já aproveitada para cada tipo.

```

SELECT
    T.id,
    T.nome,
    t.limite_horas - SUM(carga_aproveitada) AS restantes
FROM
    solicitacao_aproveitamento AS S
    INNER JOIN tipo_atividade AS T ON S.id_tipo = T.id
WHERE
    (S.matricula_aluno = @matricula
    AND S.situacao = 'Aprovada')
GROUP BY
    T.id

UNION

SELECT id, nome, limite_horas FROM tipo_atividade WHERE codigo_curso IN
(SELECT
    codigo_curso
FROM
    aluno
WHERE
    matricula = @matricula)
    AND id NOT IN (SELECT
    T.id

```

```
FROM
    solicitacao_aproveitamento AS S
    INNER JOIN tipo_atividade AS T ON S.id_tipo = T.id
WHERE
    (S.matricula_aluno = @matricula
    AND S.situacao = 'Aprovada'))
ORDER BY id;
```

17. Pensada para selecionar um supervisor que já está cadastrado no sistema. Esse supervisor será associado a uma solicitação que está sendo criada, evitando entradas duplicadas na tabela.

```
SELECT
    id
FROM
    supervisor
WHERE
    email=@email AND nome=@nome AND sobrenome=@sobrenome AND
    telefone=@telefone;
```

Nenhuma alteração foi feita nas queries 5, 6, 8, 10, 11, e 12. Todas as queries e explicações da sua serventia podem ser encontradas no arquivo [queries.sql](#).

Após a inserção de todos os dados através das migrations, a query a seguir foi executada apenas uma vez para alterar a senha de todos os usuários para “123” e os nomes das atividade para um nome genérico (no ambiente de testes).

```
UPDATE coordenador SET hash_senha=
'$2b$10$4lWiIhIeBZtSbKa5pI07xetNfkftFwT.ZfB2ahbVGeL4jJoLq9s1m';

UPDATE aluno SET hash_senha=
'$2b$10$4lWiIhIeBZtSbKa5pI07xetNfkftFwT.ZfB2ahbVGeL4jJoLq9s1m';

UPDATE tipo_atividade SET nome= 'Atividade X';
```

Indexação

Para avaliar a necessidade e efeito da adição de índices, populamos o banco de dados com entradas em números que julgamos razoáveis para o uso real da aplicação, sendo:

- 200 cursos
- 800 coordenadores e ex-coordenadores de cursos
- 3800 tipos de atividade (cerca de 20 por curso)
- 20000 alunos (cerca de 100 por curso)
- 200000 solicitações de aproveitamento (cerca de 10 por aluno)
- 50000 supervisores (um para cada quatro solicitações)
- 400000 anexos (cerca de dois por solicitação)

As tabelas *coordenador*, *curso*, *tipo_atividade* e *aluno* foram populadas através da ferramenta [SB Data Generator](#). As tabelas *supervisor*, *anexo* e *solicitacao_aproveitamento* foram populadas através de scripts em Python. Os scripts e o arquivo sql com a inserção dos dados na tabela estão disponíveis [nesta pasta](#) do Google Drive do projeto.

Escolhas dos índices

Primeiramente, é necessário levar em consideração que o PostgreSQL cria índices automaticamente nas chaves primárias das tabelas.

Nas queries 5, 6, 7, 11, 12, os índices nas chaves primárias são os únicos que seriam usados, não é necessário criar novos índices. Baseado nas outras queries, observamos que seria positivo criar os seguintes índices:

- A. *idx_solicitacao_por_aluno* em *solicitacao_aproveitamento(matricula_aluno)*, pois:
 - a. Acreditamos que irá otimizar significativamente as queries 3, 4, 10, 13, 15 e 16.
 - b. Há um número grande de solicitações de aproveitamento no banco.
 - c. O valor do campo não tem muitas repetições.
 - d. Após a criação da solicitação, seu aluno não muda, portanto não há tantas alterações custosas em comparação com o número de queries executadas beneficiadas pelo índice.
- B. *idx_anexos_por_solicitacao* em *anexo(solicitacao_id)*, pois:
 - a. Acreditamos que irá otimizar a query 8.
 - b. Há um número muito grande de anexos no banco.
 - c. O valor do campo tem também pouquíssimas repetições.
 - d. Após a criação do anexo, a sua *solicitacao_id* não muda.
- C. *idx_tipos_por_curso* em *tipo_atividade(codigo_curso)*, pois:
 - a. Apesar do número de cursos e tipos de atividade não ter uma magnitude tão grande, é raro que um novo tipo de atividade seja adicionado, portanto o custo de atualizar o índice é infrequente e baixo.
 - b. Acreditamos que irá otimizar as queries 9, 14 e 16, que são executadas com bastante frequência.
 - c. Após a criação do *tipo_atividade*, seu *codigo_curso* não muda.

- D. *idx_supervisor_unique* em *supervisor(email, nome, sobrenome, telefone)*, pois:
- Queremos evitar duplicatas em supervisor.
 - Acreditamos que irá otimizar a query 17.

O SQL utilizado para criar os índices foi:

```
CREATE INDEX idx_solicitacao_por_aluno ON
solicitacao_aproveitamento(matricula_aluno);

CREATE INDEX idx_tipos_por_curso ON tipo_atividade(codigo_curso);

CREATE INDEX idx_anexos_por_solicitacao ON anexo(solicitacao_id);

CREATE UNIQUE INDEX idx_supervisor_unique ON supervisor(email, nome,
sobrenome, telefone);
```

Apesar de potencialmente haver queries que poderiam se beneficiar, optamos por não criar índices em:

- curso(matricula_coordenador)*, pois embora seja relevante para as queries 1 e 2, o número de cursos é relativamente baixo e a indexação teria pouco impacto.
- solicitacao_aproveitamento(situacao)*, pois embora seja relevante para as queries 2, 4, 10, 13, 15 e 16, há pouca variedade de valores nesse campo ('Aprovada', 'Reprovada' ou 'Pendente').
- tipo_atividade(tipo_carga_horaria)* pois há pouca variedade de valores nesse campo ('G' ou 'E').
- solicitacao_aproveitamento(id_tipo)*, pois só seria utilizada em uma das queries (15) e tornaria ainda mais custosa a inserção de novos registros em *solicitacao_aproveitamento*, onde já criamos um índice.

Resultados

Executamos algumas queries e inserções com ``EXPLAIN (ANALYZE TRUE, TIMING FALSE)``, calculando uma média do tempo de execução antes e depois da criação de cada índice. Um exemplo:

Query 8 antes da criação de *idx_anexos_por_solicitacao*:


```

1 EXPLAIN (ANALYZE TRUE, TIMING FALSE) SELECT * FROM anexo
2 WHERE solicitacao_id = 111111;

```

QUERY PLAN

Index Scan using anexo_pkey on anexo (cost=...
Index Cond: (solicitacao_id = 111111)
Planning Time: 0.057 ms
Execution Time: 3.665 ms

Query 8 após a criação de *idx_anexos_por_solicitacao*:

```

1 CREATE INDEX idx_anexos_por_solicitacao ON anexo(solicitacao_id);
2
3 EXPLAIN (ANALYZE TRUE, TIMING FALSE) SELECT * FROM anexo
4 WHERE solicitacao_id = 111111;

```

QUERY PLAN

Index Scan using idx_anexos_por_solicitacao o...
Index Cond: (solicitacao_id = 111111)
Planning Time: 0.057 ms
Execution Time: 0.023 ms

Obtivemos os seguintes resultados de tempo de execução das queries para os índices *idx_solicitacao_por_aluno* (A), *idx_anexos_por_solicitacao* (B), *idx_tipos_por_curso* (C) e *idx_supervisor_unique* (D).

ÍNDICE:	A		B		C		D	
	ANTES	DEPOIS	ANTES	DEPOIS	ANTES	DEPOIS	ANTES	DEPOIS
Query 8			3,5 ms	0,025 ms				
Query 9					0,3 ms	0,30 ms		
Query 10	50 ms	0,1 ms						
Query 13	50 ms	0,1 ms						
Query 14					0,32 ms	0,04 ms		
Query 16	130 ms	0,8 ms			130 ms	130 ms		
Query 17							3,4 ms	0,04 ms
INSERT	0,07 ms	0,1 ms	0,1 ms	0,15 ms	0,07 ms	0,08 ms	0,03 ms	0,05 ms

Como o impacto negativo no tempo de inserção não foi muito significativo após a criação, e todos os índices causaram alguma aceleração significativa das queries, escolhemos

manter os quatro índices sugeridos na versão final do banco de dados. Os índices estão definidos nas últimas linhas do arquivo [create.sql](#).

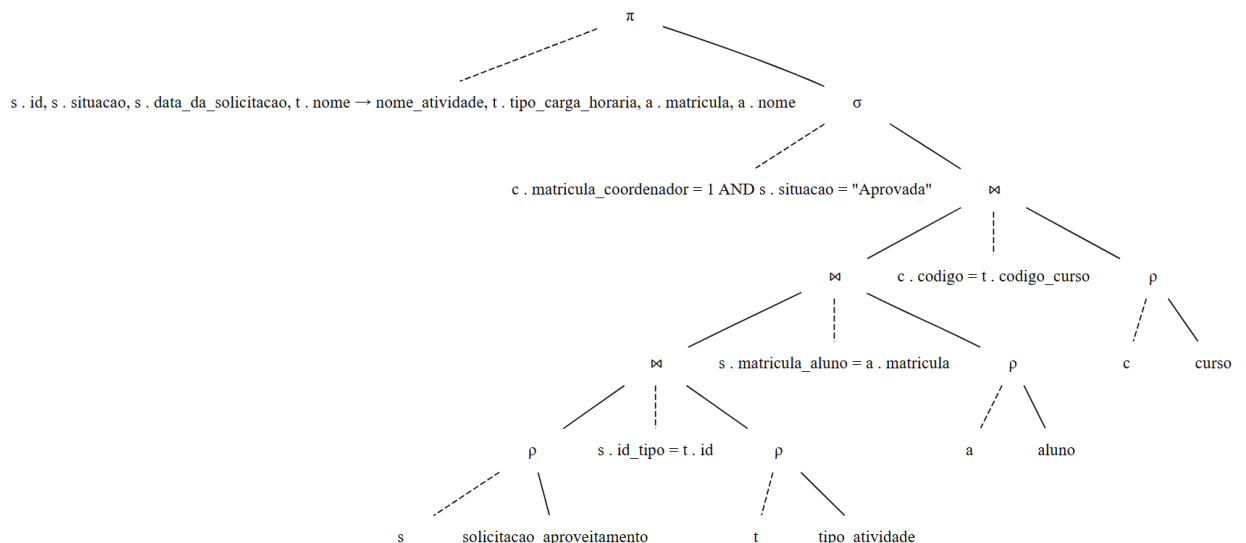
Otimização

A metodologia que empregamos para analisar as otimizações feitas pelo query planner do SGBD PostgreSQL consta em construir a árvore de consulta original na álgebra relacional, usar o comando **EXPLAIN VERBOSE** para construir uma árvore de consulta otimizada e analisar as mudanças feitas, justificando por que otimizam a execução das queries. Faremos isso para duas queries específicas.

a) Query 2:

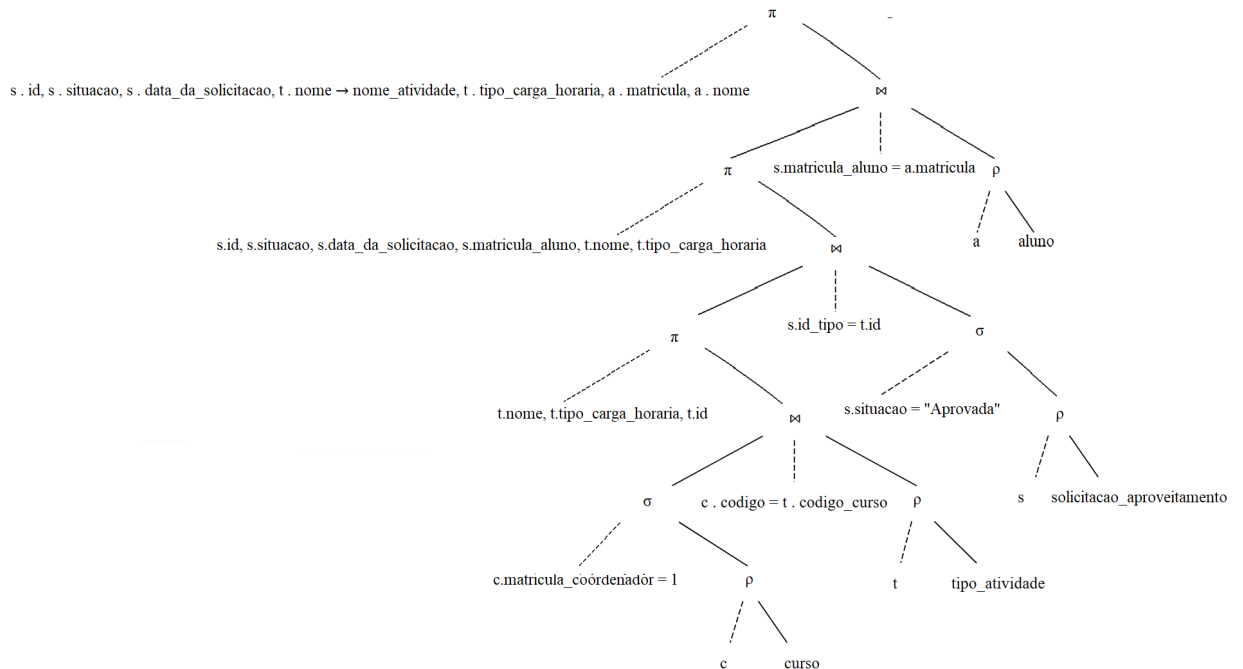
```
SELECT S.id, S.situacao, S.data_da_solicitacao, T.nome AS nome_atividade,  
      T.tipo_carga_horaria, A.matricula, A.nome  
FROM  
      solicitacao_aproveitamento AS S  
      INNER JOIN tipo_atividade AS T ON S.id_tipo = T.id  
      INNER JOIN aluno AS A ON S.matricula_aluno = A.matricula  
      INNER JOIN curso AS C ON C.codigo = T.codigo_curso  
WHERE C.matricula_coordenador = @matricula AND S.situacao = @situacao;
```

Árvore de consulta original da query 2:



Resultado do comando EXPLAIN VERBOSE na query 2:

```
Gather (cost=1058.42..24270.38 rows=333 width=75)
Output: s.id, s.situacao, s.data_da_solicitacao, t.nome, t.tipo_carga_horaria, a.matricula, a.nome
Workers Planned: 2
-> Nested Loop (cost=58.42..23237.08 rows=139 width=75)
Output: s.id, s.situacao, s.data_da_solicitacao, t.nome, t.tipo_carga_horaria, a.matricula, a.nome
Inner Unique: true
-> Hash Join (cost=58.14..23190.18 rows=139 width=69)
Output: s.id, s.situacao, s.data_da_solicitacao, s.matricula_aluno, t.nome, t.tipo_carga_horaria
Hash Cond: (s.id_tipo = t.id)
-> Parallel Seq Scan on atividades_complementares.solicitacao_aproveitamento s (cost=0.00..23026.70 rows=27723 width=25)
Output: s.id, s.descricao, s.resposta_coordenador, s.data_da_solicitacao, s.situacao, s.carga_real, s.carga_aproveitada, s.matricula_aluno, s.matricula_coordenador, s.id_tipo, s.id_supervisor
Filter: ((s.situacao)::text = 'Aprovada'::text)
-> Hash (cost=57.90..57.90 rows=19 width=52)
Output: t.nome, t.tipo_carga_horaria, t.id
-> Nested Loop (cost=4.43..57.90 rows=19 width=52)
Output: t.nome, t.tipo_carga_horaria, t.id
-> Seq Scan on atividades_complementares.curso c (cost=0.00..6.50 rows=1 width=4)
Output: c.codigo, c.nome, c.horas_extensao, c.horas_gerais, c.matricula_coordenador
Filter: (c.matricula_coordenador = 1)
-> Bitmap Heap Scan on atividades_complementares.tipo_atividade t (cost=4.43..51.21 rows=19 width=56)
Output: t.id, t.nome, t.tipo_carga_horaria, t.limite_horas, t.horas, t.requer_supervisor, t.codigo_curso
Recheck Cond: (t.codigo_curso = c.codigo)
-> Bitmap Index Scan on idx_tipos_por_curso (cost=0.00..4.42 rows=19 width=0)
Index Cond: (t.codigo_curso = c.codigo)
-> Index Scan using aluno_pkey on atividades_complementares.aluno a (cost=0.29..0.34 rows=1 width=10)
Output: a.matricula, a.cpf, a.telefone, a.nome, a.sobrenome, a.email, a.hash_senha, a.codigo_curso
Index Cond: (a.matricula = s.matricula_aluno)
```



Notamos que a seleção que tinha duas condições em **AND** na query original foi dividida em seleções independentes para que pudessem ser aplicadas o mais cedo possível. Isso reduz o número de linhas a serem processadas nos passos posteriores. Além disso, projeções

também são feitas em etapas intermediárias ao invés de apenas ao final, reduzindo o espaço ocupado pelos resultados temporários na memória.

Também foi alterada a ordem dos **INNER JOINS**. Como seus atributos não são usados nas condições de outros **INNER JOIN** ou seleções, o **INNER JOIN** com a tabela aluno foi deixado por último, depois da projeção feita sobre outros **INNER JOINS**, reduzindo o uso de memória.

b) Query 7:

SELECT

SA.*,

S.nome AS nome_supervisor, S.sobrenome AS sobrenome_supervisor,

S.email AS email_supervisor, S.telefone AS tel_supervisor,

T.nome AS nome_atividade, T.tipo_carga_horaria, T.limite_horas, T.horas,

A.nome AS nome_aluno, A.sobrenome AS sobrenome_aluno,

C.nome AS nome_coordenador, C.sobrenome AS sobrenome_coordenador

FROM

solicitacao_aproveitamento AS SA

INNER JOIN supervisor AS S ON SA.id_supervisor = S.id

INNER JOIN tipo_atividade AS T ON SA.id_tipo = T.id

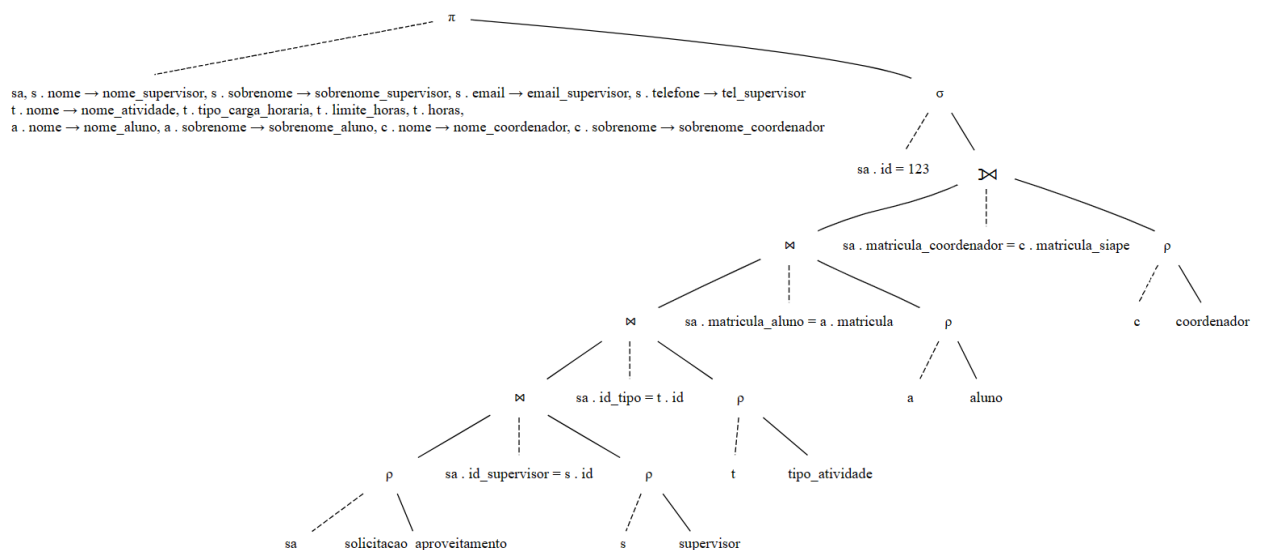
INNER JOIN aluno AS A ON SA.matricula_aluno = A.matricula

LEFT OUTER JOIN coordenador AS C ON SA.matricula_coordenador =

C.matricula_siape

WHERE SA.id = @solicitacao;

Árvore de consulta original da query 7:



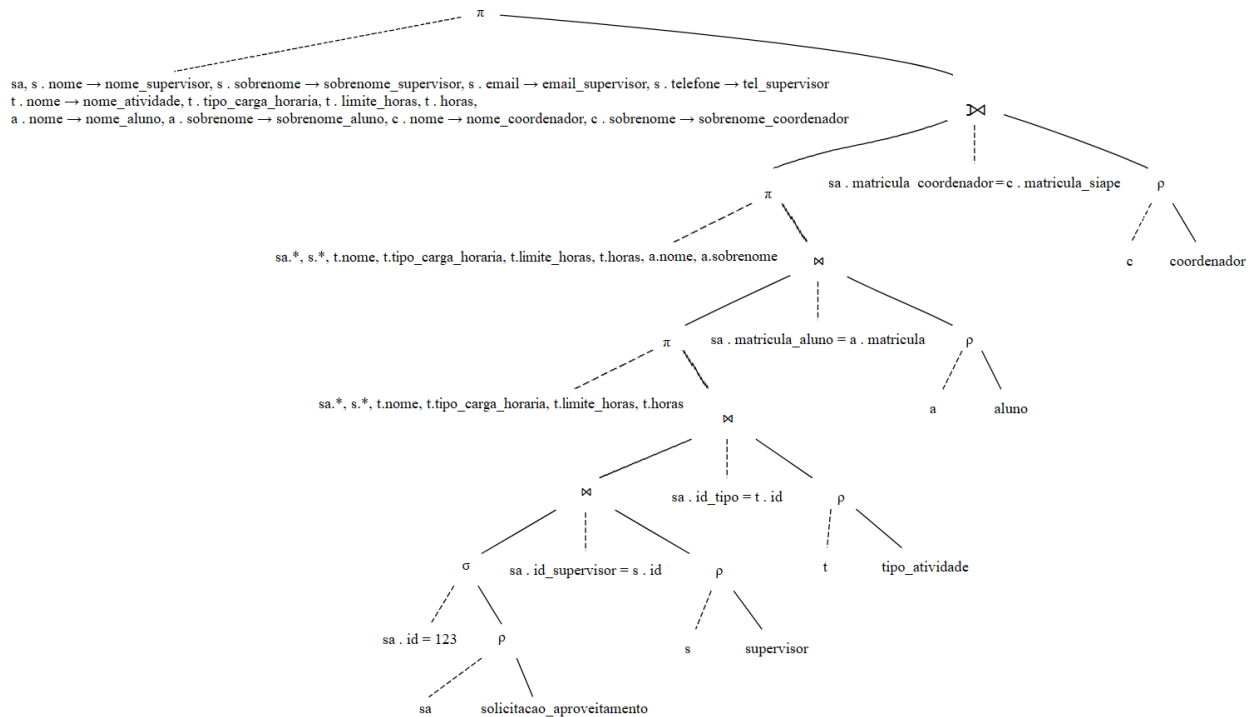
Resultado do comando EXPLAIN VERBOSE na query 7:

```

QUERY PLAN
Nested Loop Left Join (cost=1.55..41.67 rows=1 width=1051)
  Output: sa.id, sa.descricao, sa.resposta_coordenador, sa.data_da_solicitacao, sa.situacao, sa.carga_real, sa.carga_aproveitada, sa.matricula_aluno, sa.matricula_coord...
  Inner Unique: true
-> Nested Loop (cost=1.28..33.35 rows=1 width=1038)
  Output: sa.id, sa.descricao, sa.resposta_coordenador, sa.data_da_solicitacao, sa.situacao, sa.carga_real, sa.carga_aproveitada, sa.matricula_aluno, sa.matricula_c...
  Inner Unique: true
-> Nested Loop (cost=0.99..25.05 rows=1 width=1025)
  Output: sa.id, sa.descricao, sa.resposta_coordenador, sa.data_da_solicitacao, sa.situacao, sa.carga_real, sa.carga_aproveitada, sa.matricula_aluno, sa.matricul...
  Inner Unique: true
-> Nested Loop (cost=0.71..16.75 rows=1 width=969)
  Output: sa.id, sa.descricao, sa.resposta_coordenador, sa.data_da_solicitacao, sa.situacao, sa.carga_real, sa.carga_aproveitada, sa.matricula_aluno, sa.mat...
  Inner Unique: true
-> Index Scan using solicitacao_aproveitamento_pkey on atividades_complementares.solicitacao_aproveitamento sa (cost=0.42..8.44 rows=1 width=915)
  Output: sa.id, sa.descricao, sa.resposta_coordenador, sa.data_da_solicitacao, sa.situacao, sa.carga_real, sa.carga_aproveitada, sa.matricula_aluno, sa...
  Index Cond: (sa.id = 1)
-> Index Scan using supervisor_pkey on atividades_complementares.supervisor s (cost=0.29..8.31 rows=1 width=58)
  Output: s.id, s.email, s.nome, s.sobrenome, s.telefone
  Index Cond: (s.id = sa.id_supervisor)
-> Index Scan using tipo_atividade_pkey on atividades_complementares.tipo_atividade t (cost=0.28..8.30 rows=1 width=60)
  Output: t.id, t.nome, t.tipo_carga_horaria, t.limite_horas, t.horas, t.requer_supervisor, t.codigo_curso
  Index Cond: (t.id = sa.id_tipo)
-> Index Scan using aluno_pkey on atividades_complementares.aluno a (cost=0.29..8.30 rows=1 width=17)
  Output: a.matricula, a.cpf, a.telefone, a.nome, a.sobrenome, a.email, a.hash_senha, a.codigo_curso
  Index Cond: (a.matricula = sa.matricula_aluno)
-> Index Scan using coordenador_pkey on atividades_complementares.coordenador c (cost=0.28..8.29 rows=1 width=17)
  Output: c.matricula_siape, c.nome, c.sobrenome, c.hash_senha, c.email
  Index Cond: (c.matricula_siape = sa.matricula_coordenador)

```

Árvore de consulta otimizada da query 7:



Assim como na otimização da query 2, a seleção é feita o mais cedo possível. Como a seleção é de uma única chave primária de solicitacao_aproveitamento, reduz-se de centenas de milhares de solicitações a serem processadas nas próximas etapas para uma única solicitação, uma otimização com impacto enorme.

Também como na otimização da query 2, projeções são feitas assim que possível nas etapas intermediárias, reduzindo o uso de memória dos resultados parciais.

Transações

As principais situações na aplicação que envolvem inserções e atualizações no banco de dados são:

1. O cadastro de um novo coordenador ou um novo aluno (pelo próprio usuário).
2. A criação de um novo curso (pelos administradores da aplicação).
3. A atribuição de um coordenador a um curso (pelos administradores).
4. A criação, exclusão ou alteração dos dados de um tipo de atividade (pelo coordenador).
5. A exclusão de uma solicitação de aproveitamento (pelo aluno).
6. A abertura (criação) de uma nova solicitação de aproveitamento (pelo usuário), que envolve o salvamento e inserção dos seus anexos, tal como a atribuição a um supervisor existente ou novo no banco.

As operações número 1, 2, 3, 4 e 5 são executadas através de um único comando. Por esse motivo, avaliamos que não é necessário definir transações para sua execução.

No caso da operação 6 é necessário garantir que a solicitação só será criada caso seja encontrado um supervisor (ou criado um novo com sucesso) e os anexos sejam inseridos no banco com sucesso. Caso a solicitação não seja inserida com sucesso, é também necessário garantir que seus anexos associados e supervisor (caso um novo tenha sido criado) não permaneçam no banco.

Para a operação 6, definimos a seguinte transação:

```
START TRANSACTION;
do $$
DECLARE solicitacao_inserida integer ;

begin

INSERT INTO supervisor (email, nome, sobrenome, telefone)
VALUES ('@email_sup', '@nome_sup', '@sobrenome_sup', '@telefone_sup')
```

```

ON CONFLICT DO NOTHING;

INSERT INTO solicitacao_aproveitamento (descricao, resposta_coordenador,
data_da_solicitacao,situacao,carga_real,carga_aproveitada,
matricula_aluno,matricula_coordenador,id_tipo,id_supervisor)
VALUES ( '@descricao', NULL, DEFAULT, 'Pendente', @carga_real,
        NULL, @matricula_aluno,@matricula_coordenador, @id_tipo,
        (SELECT id FROM supervisor WHERE email = '@email_sup' and
         nome = '@nome_sup' and
         sobrenome = '@sobrenome_sup' and
         telefone = '@telefone_sup'
        )
        )
        )returning id into solicitacao_inserida;

if @nome_anexo1!='' then
INSERT INTO anexo
VALUES
    (
        1, solicitacao_inserida, '@nome_anexo1',
        '@extensao_anexo1', '@caminho_anexo1'
    );
end if;

if @nome_anexo2!='' then
INSERT INTO anexo
VALUES
    (
        2, solicitacao_inserida, '@nome_anexo2',
        '@extensao_anexo2', '@caminho_anexo2'
    );
end if;

if @nome_anexo3!='' then
INSERT INTO anexo
VALUES
    (
        3, solicitacao_inserida, '@nome_anexo3',
        '@extensao_anexo3', '@caminho_anexo3'
    );
end if;

```

```
end $$;  
COMMIT;
```

Backup

Optamos por fazer backup diário do banco de dados por considerar que, no cenário de uma perda de dados, é um período curto o suficiente para minimizar os danos causados a meras inconveniências.

Caso aconteça de as solicitações cadastradas em um dia serem perdidas antes de ser realizado backup, os alunos que as cadastraram muito provavelmente ainda têm em sua posse as informações e arquivos que possibilitaram criá-las, e portanto são capazes de refazê-las.

Concluímos que o melhor horário para realizar o backup seria durante a madrugada, tendo em vista que é um sistema para alunos e professores da UFBA, em geral localizados no mesmo fuso-horário e previsivelmente haverá pouquíssima atividade durante esse horário.

Para realizar o backup e restore do banco de dados PostgreSQL utilizaremos o utilitário para dump de imagem do banco *pg_dump*, incluído por padrão em todas as instalações do PostgreSQL. Tomamos essa decisão pois é uma solução simples, automatizável e confiável. Embora o *pg_dump* seja capaz de fazer *hot backup*, por uma melhor garantia de consistência optamos por sempre desligar a aplicação durante os backups, garantindo que não há operações de update ou insert sendo executadas.

Após gerar o arquivo de dump, planejamos armazená-los localmente e também em nuvem por 30 dias. É importante manter os backups em duas localizações diferentes, pois evita que um acidente ou catástrofe em uma localidade cause a perda total de todos os backups da aplicação.

Para executar o backup é necessário primeiramente encerrar o back-end da aplicação, cessando os acessos ao banco de dados. A seguir, executa-se os passos descritos abaixo:

No Windows

Localizar a pasta de instalação do PostgreSQL. No nosso caso o caminho para pasta é:

```
C:\Program Files\PostgreSQL\15\bin
```

Após isso é preciso executar o terminal do Windows como administrador e executar o comando `cd` com o caminho da pasta, como no exemplo abaixo:


```
cd C:\Program Files\PostgreSQL\15\bin
```

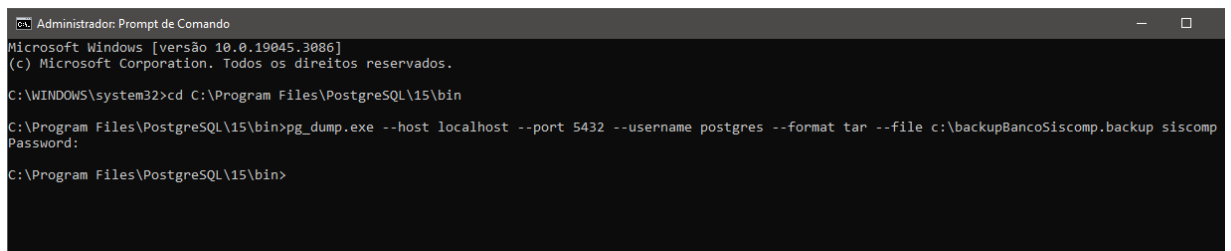
Fazendo o Backup:

Executar o comando:

```
pg_dump.exe --host localhost --port 5432 --username postgres  
--format tar --file C:\backupBancoSiscomp.backup siscomp
```

- *--host localhost*: define o endereço IP para a conexão com o banco, pode ser local ou externo em outra rede;
- *--port 5432*: define a porta utilizada para a conexão, nesse caso a padrão PostgreSQL 5432;
- *--username postgres*: define qual é o usuário utilizado na comunicação;
- *--format tar*: o tipo de compressão do arquivo gerado;
- *--file C:\backupBancoSiscomp.backup*: define o nome e caminho completo do arquivo que será gerado;
- *siscomp*: o nome no PostgreSQL do banco de dados da aplicação

Será pedida a senha de acesso. Após informá-la, o arquivo será gerado.

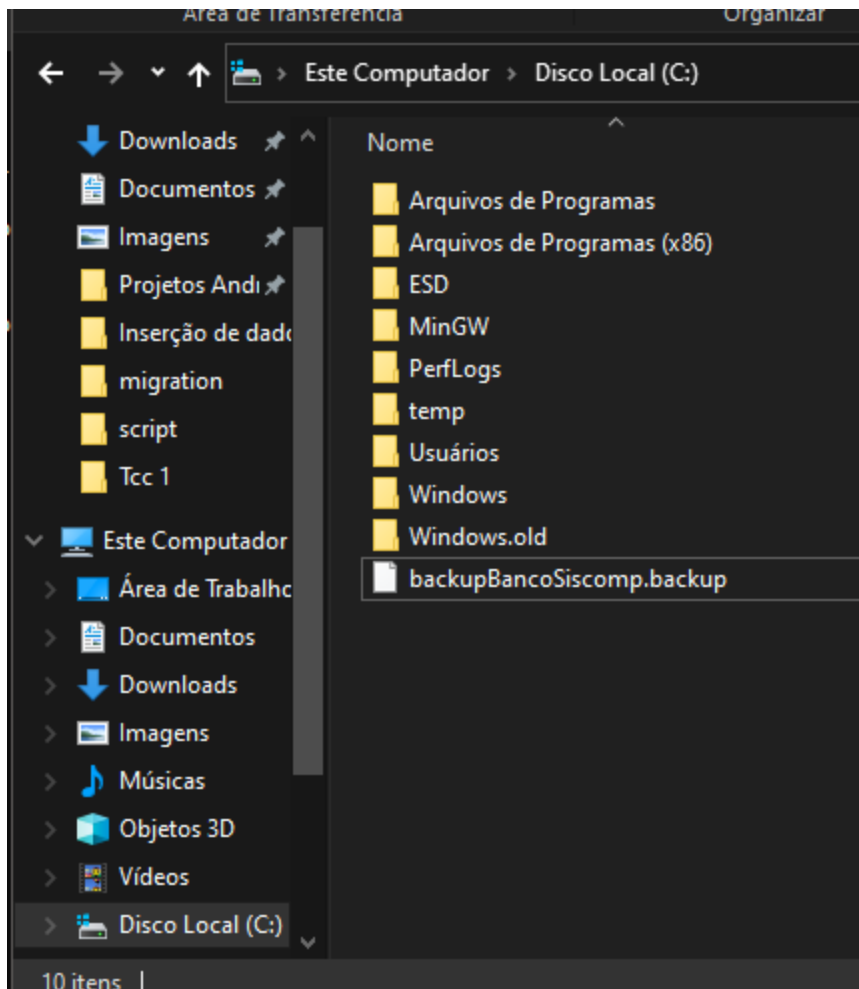


```
Administrador: Prompt de Comando
Microsoft Windows [versão 10.0.19045.3086]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\WINDOWS\system32>cd C:\Program Files\PostgreSQL\15\bin

C:\Program Files\PostgreSQL\15\bin>pg_dump.exe --host localhost --port 5432 --username postgres --format tar --file c:\backupBancoSiscomp.backup siscomp
Password:

C:\Program Files\PostgreSQL\15\bin>
```



Fazendo a Restauração:

Executar o comando:

```
pg_restore.exe --host localhost --port 5432 --username postgres  
--dbname siscomp c:\backupBancoSiscomp.backup
```

- *--host localhost*: define o endereço IP para a conexão com o banco, pode ser local ou externo em outra rede;
- *--port 5432*: define a porta utilizada para a conexão, nesse caso a padrão PostgreSQL 5432;
- *--username postgres*: define qual é o usuário utilizado na comunicação;
- *--dbname siscomp*: o nome no PostgreSQL do banco de dados da aplicação
- *C:\backupBancoSiscomp.backup*: O caminho completo do arquivo que deseja restaurar

Será pedida a senha de acesso ao banco. Após informá-la, o banco de dados será restaurado.

```
C:\Program Files\PostgreSQL\15\bin>pg_restore.exe --host localhost --port 5432 --username postgres --dbname siscomp c:\backupBancoSiscomp.backup
Password:
C:\Program Files\PostgreSQL\15\bin>
```

Linux

Fazendo o Backup:

`pg_dump -U postgres -W -F t siscomp > backupBancoSiscomp.tar`

- *-U postgres*: especifica o usuário para se conectar ao servidor de banco de dados PostgreSQL. Usamos o postgres neste exemplo.
- *-W*: força o *pg_dump* a solicitar a senha antes de conectar ao servidor de banco de dados. Depois de pressionar Enter o *pg_dump* solicitará a senha do usuário PostgreSQL.
- *-F t*: especifica o formato do arquivo de saída (tar).
- *siscomp*: é o nome do banco de dados do qual deseja fazer backup.

Fazendo a restauração:

`pg_restore -d siscomp backupBancoSiscomp.tar`

Após este procedimento pode-se reiniciar o back-end da aplicação e retornar ao funcionamento normal. Todo esse processo pode ser automatizado através de scripts que serão executados em um horário específico na madrugada.